A clique-based method using dynamic programming for computing edit distance between unordered trees.

AUTHOR(S):

Mori, Tomoya; Tamura, Takeyuki; Fukagawa, Daiji; Takasu, Atsuhiro; Tomita, Etsuji; Akutsu, Tatsuya

CITATION:

Mori, Tomoya ...[et al]. A clique-based method using dynamic programming for computing edit distance between unordered trees.. Journal of computational biology 2012, 19(10): 1089-1104

ISSUE DATE:

2012-10

A Clique-Based Method Using Dynamic Programming for Computing Edit Distance between Unordered Trees

Tomoya Mori[1], Takeyuki Tamura[1], Daiji Fukagawa[2], Atsuhiro Takasu[3], Etsuji Tomita[4], Tatsuya Akutsu[1,*]

[1] Bioinformatics Center, Institute for Chemical Research, Kyoto University, Kyoto, Japan

[2] Faculty of Culture and Information Science, Doshisha University, Kyoto, Japan

[3] National Institute of Informatics, Tokyo, Japan

[4] University of Electro-Communications, Tokyo, Japan

[*] Corresponding Author: takutsu @ kuicr.kyoto-u.ac.jp

Abstract

Many kinds of tree structured data such as RNA secondary structures have become available due to the progress of techniques in the field of molecular biology. In order to analyze the tree structured data, various measures for computing the similarity between them have been developed and applied. Among them, tree edit distance is one of the most widely used measures. However, the tree edit distance problem for unordered trees is NP-hard. Therefore, it is required to develop efficient algorithms for the problem. Recently, a practical method called clique-based algorithm has been proposed but it is not fast for large trees.

This paper presents an improved clique-based method for the tree edit distance problem for unordered trees. The improved method is obtained by introducing a dynamic programming scheme and heuristic techniques to the previous clique-based method. In order to evaluate the efficiency of the improved method, we applied the method to comparison of real tree structured data such as glycan structures. For large tree structures, the improved method is much faster than the previous method. In particular, for hard instances, the improved method achieved more than 100 times speed-up.

Key words: unordered trees, tree edit distance, maximum clique, dynamic programming, glycan

1 INTRODUCTION

Tree structured data such as RNA secondary structures (Jiang et al., 2002; Zaki et al., 2005), phylogenetic trees (Horesh et al., 2006), glycans (Aoki et al., 2004), and vascular trees (Yu et al., 2007) often appear in computational biology. Consequently, various techniques have been developed and applied to analysis of these tree structured data. Among them, comparison of tree structured data is important because it can be used for searching for similar objects. The *tree edit distance* is one of the most widely used measures for comparison of tree structured data (Bille, 2005). In this measure, the distance between two trees is measured by the minimum cost sequence of edit operations that transforms one tree into another tree where an edit operation is either a *deletion* of a node, an *insertion* of a node, or a *substitution* of a label of a node. For the tree edit distance problem for ordered trees, Tai developed an $O(n^6)$ time algorithm (Tai, 1979), where n is the number of nodes in a larger input tree. After several improvements, Demaine et al. developed an $O(n^3)$ time algorithm and showed that this bound is optimal under some computation strategy (Demaine et al., 2009).

The tree edit distance between ordered trees is useful if the ordering among children has important meanings. However, it is preferable to regard input trees as unordered trees in some applications (Aoki et al., 2004; Horesh et al., 2006). Unfortunately, Zhang et al. proved that the tree edit distance problem for unordered trees is NP-hard (Zhang et al., 1992). In order to cope with this hardness, Akutsu et al. developed a fixed parameter algorithm which works in $O(2.62^k \cdot poly(n))$ time (Akutsu et al., 2011a), where k is the maximum allowed edit distance. Although their algorithm might be useful for comparison of very similar trees (i.e., where k is small), it is not useful for comparison of non-similar trees. Horesh et al. developed an A^* algorithm (Horesh et al., 2006). Although their algorithm works efficiently for comparison of moderate size unlabeled trees under the unit cost distance (i.e., the cost of each edit operation is 1), it is unclear whether it can be efficiently applied to labeled trees or general cost cases.

Fukagawa et al. recently proposed a practical method for computing the tree edit distance between unordered trees (Fukagawa et al., 2011) using algorithms for computing the *maximum clique* (Tomita et al., 2010, 2011). In this method, an instance of the tree edit distance is directly transformed into an instance of the *maximum vertex weighted clique* problem and then an existing clique solver (Nakamura and Tomita, 2005) is applied. Although similar reductions have been proposed for variants of the tree edit distance problem (Pelillo et al., 1999; Torsello and Hancock, 2003) and other problems (Ogawa, 1986), to the best of our knowledge, it was the first method that exactly solves the proper tree edit distance problem for unordered trees using the maximum clique. The method was applied to comparison and search of similar glycan structures and shown to be efficient for moderate size tree structures (Fukagawa et al., 2011). However, it was not fast enough if large glycan or tree structures were given.

Therefore, in the preliminary version of this paper (Akutsu et al., 2011b), we improved the method of (Fukagawa et al., 2011) and developed a *dynamic programming*-based (DP) algorithm that repeatedly solves instances of the maximum vertex weighted clique prob-

lem as sub-problems. Due to this improvement, sparser graphs are generated and thus maximum clique instances can be solved more efficiently in many cases. Although multiple clique instances must be solved in the improved method, it is expected that speed up due to sparsity is more beneficial if input trees are large. Furthermore, by utilizing the feature of DP, we introduced heuristic techniques which do not violate the optimality of the solution. When it was applied to comparison of large glycan structures, our improved method showed speed-up in most cases. However, there still exist cases for which it takes long CPU time. In particular, it takes very long CPU time if there exist many leaves. In such a case, constructed graphs would contain many vertices and edges and thus a clique algorithm does not work efficiently.

In this full version, we augment this DP-based approach by introducing new heuristic techniques to further reduce the computation time without violating the optimality of the solution, especially for trees with many leaves or many isomorphic subtrees. Furthermore, in order to utilize maximum clique algorithms in place of maximum vertex weighted clique algorithms, we develop a new clique-based method for computing the unordered tree edit distance in which the maximum vertex weighted clique problem is transformed into the maximum clique problem.

We compare the improved clique-based method and the maximum clique-based method with the previous maximum vertex weighted clique-based method (Fukagawa et al., 2011) using glycan data obtained from the KEGG database (Kanehisa et al., 2010) and Web logs data CSLOGS opened to the public[1] (Zaki et al., 2005). The results suggest that the improved clique-based method is much faster than the maximum clique-based method and the previous clique-based method (Fukagawa et al., 2011) in most cases of comparison of large tree structured data. In particular, when there exist many leaves or isomorphic subtrees, our improved method shows significant speed-up.

[1] http://www.cs.rpi.edu/~zaki/software/

2 TREE EDIT DISTANCE

Before presenting the method, we briefly review *tree edit distance* and *edit distance mapping* for rooted, labeled, and unordered trees (Bille, 2005; Zhang et al., 1992).

Let T be a rooted unordered tree where each node v has a label $\ell(v)$ over an alphabet Σ. $r(T)$, $V(T)$, and $E(T)$ denote the root, the set of nodes, and the set of edges of T, respectively. For a node $v \in V(T)$, $des(v)$ and $T(v)$ denote the set of descendants of v (not including v) and the subtree induced by v and its descendants, respectively. In this paper, n denotes the number of nodes in a larger input tree, that is, $n = \max\{|V(T_1)|, |V(T_2)|\})$ where T_1 and T_2 are input trees.

An *edit operation on a tree* T is either a *deletion*, an *insertion*, or a *substitution*, each of which is defined by (see also Fig. 1):

- **Deletion**: Delete a non-root node v in T with parent u, making the children of v become children of u. The children are inserted in the place of v into the set of the children of u.
- **Insertion**: Inverse of the deletion. Insert a node v as a child of u in T, making v the parent of some of the children of u.
- **Substitution**: Change the label of a node v in T.

(Figure 1)

For each edit operation, the *cost* is defined as follows:

- $\gamma(a, \epsilon)$: cost of deleting a node labeled with a.
- $\gamma(\epsilon, a)$: cost of inserting a node labeled with a,
- $\gamma(a, b)$: cost of substituting a node with label a to label b,

The *edit distance* $dist(T_1, T_2)$ between two unordered trees T_1 and T_2 is the cost of the minimum cost sequence of edit operations that transforms T_1 to T_2, where we adopt the following standard assumption so that $dist(T_1, T_2)$ becomes a distance metric (Bille, 2005; Zhang et al., 1992):

- $\gamma(a, b) \geq 0$ for any $(a, b) \in \Sigma' \times \Sigma'$,
- $\gamma(a, a) = 0$ for any $a \in \Sigma'$,
- $\gamma(a, b) = \gamma(b, a)$ for any $(a, b) \in \Sigma' \times \Sigma'$,
- $\gamma(a, c) \leq \gamma(a, b) + \gamma(b, c)$ for any $(a, b, c) \in \Sigma' \times \Sigma' \times \Sigma'$,

where $\Sigma' = \Sigma \cup \{\epsilon\}$.

There is a close relationship between the edit distance and the *edit distance mapping* (or just *mapping*) (Bille, 2005; Zhang et al., 1992). $M \subseteq V(T_1) \times V(T_2)$ is called a *mapping* if the following conditions are satisfied for any two pairs (u_1, v_1), $(u_2, v_2) \in M$ (see also Fig. 2:

(i) $u_1 = u_2$ iff $v_1 = v_2$,

(ii) $u_1 \in des(u_2)$ iff $v_1 \in des(v_2)$.

(Figure 2)

Let I_1 and I_2 be the sets of nodes in $V(T_1)$ and $V(T_2)$ not appearing in M, respectively. Then, the following equality holds (Bille, 2005; Zhang et al., 1992):

$$dist(T_1, T_2) = \min_M \{ \sum_{u \in I_1} \gamma(\ell(u), \epsilon) + \sum_{v \in I_2} \gamma(\epsilon, \ell(v)) + \sum_{(u,v) \in M} \gamma(\ell(u), \ell(v)) \}.$$

Here we introduce a *score function* $f(u, v)$ for $(u, v) \in V(T_1) \times V(T_2)$ defined by

$$f(u, v) = \gamma(\ell(u), \epsilon) + \gamma(\epsilon, \ell(v)) - \gamma(\ell(u), \ell(v)).$$

Then, we can see that $f(u, v) = f(v, u) \geq 0$ holds. It should also be noted that under the unit cost model (i.e., $\gamma(a, b) = 1$ for all $a \neq b$), $f(u, v) = 2$ holds if $\ell(u) = \ell(v)$, and $f(u, v) = 1$ holds otherwise. Let $score(M)$ be the score of a mapping M defined by

$$score(M) = \sum_{(u,v) \in M} f(u, v).$$

Let M_{OPT} be the mapping with the maximum score. Then, the following property holds (Akutsu et al., 2011a):

$$\begin{aligned}
dist(T_1, T_2) &= \min_M \{ \sum_{u \in I_1} \gamma(\ell(u), \epsilon) + \sum_{v \in I_2} \gamma(\epsilon, \ell(v)) + \sum_{(u,v) \in M} \gamma(\ell(u), \ell(v)) \} \\
&= \min_M \{ \sum_{u \in V(T_1)} \gamma(\ell(u), \epsilon) + \sum_{v \in V(T_2)} \gamma(\epsilon, \ell(v)) \\
&\qquad + \sum_{(u,v) \in M} (\gamma(\ell(u), \ell(v)) - \gamma(\ell(u), \epsilon) - \gamma(\epsilon, \ell(v))) \} \\
&= \sum_{u \in V(T_1)} \gamma(\ell(u), \epsilon) + \sum_{v \in V(T_2)} \gamma(\epsilon, \ell(v)) \\
&\qquad - \max_M \{ \sum_{(u,v) \in M} (\gamma(\ell(u), \epsilon) + \gamma(\epsilon, \ell(v)) - \gamma(\ell(u), \ell(v))) \} \\
&= \sum_{u \in V(T_1)} \gamma(\ell(u), \epsilon) + \sum_{v \in V(T_2)} \gamma(\epsilon, \ell(v)) - score(M_{OPT}), \qquad (1)
\end{aligned}$$

assuming that the root of T_1 corresponds to the root of T_2 in M_{OPT}, where this assumption can be removed if we add dummy nodes as new roots. It is to be noted that the first and second terms in the right hand side of the last equality are invariant with a mapping. Therefore, this equality means that the tree edit distance can be obtained by computing a mapping with the maximum score.

3 METHOD

3.1 Maximum Vertex Weighted Clique

Let $G = (V, E)$ be an undirected graph. A subgraph $G' = (V', E')$ of $G = (V, E)$ is called a *clique* if it is a complete subgraph (i.e., $\{\{v_i, v_j\} \mid v_i, v_j \in V',\ v_i \neq v_j\} = E'$). The *maximum clique* problem is to find a clique with the maximum number of vertices in a given undirected graph $G = (V, E)$. Although the maximum clique problem is NP-hard, several practical algorithms have been developed (Tomita et al., 2010, 2011). In this paper, we use a variant of the maximum clique problem called *the maximum vertex weighted clique problem*. In this variant, each vertex v has a weight $w(v)$ and the problem is to find a clique $G' = (V', E')$ which maximizes $\sum_{v \in V'} w(v)$ (see also Fig. 3).

(Figure 3)

3.2 Algorithm MWCQ and MCS

Nakamura and Tomita developed a practically efficient algorithm called MWCQ for the maximum vertex weighted clique problem (Nakamura and Tomita, 2005). After preliminary experiments on maximum vertex weighted clique algorithms (Nakamura and Tomita, 2005), we employ MWCQ as a solver for the maximum vertex weighted clique problem. Here, we briefly review MWCQ.

The underlining algorithm of MWCQ is a very simple and fast branch-and-bound depth-first-search algorithm MCQ for finding a maximum clique of an unweighted graph (Tomita and Seki, 2003; Tomita et al., 2011). MCQ employs greedy approximate coloring to obtain an upper bound of the size of a maximum clique. The size of a maximum clique in an unweighted graph is bounded above by the number of approximate color classes (the total number of disjoint sets of independent set). This relation contributes to an effective bounding condition.

For a vertex weighted graph, the maximum weight of a clique in a graph is bounded above by the summation of the maximum weight in each approximate color class (independent set). Then we have a simple algorithm MWCQ for finding a maximum vertex weighted clique by introducing the above new bounding condition into MCQ instead of the previous one together with appropriate ordering of vertices as in MCQ.

Furthermore, Tomita et al. proposed a new branch-and-bound algorithm MCS for the maximum clique problem (Tomita et al., 2010). In MCS, new approximate coloring is introduced along with other new techniques, which makes MCS much faster than MCQ for most instances. In order to utilize MCS, to be shown below, we develop a method which does not use a maximum vertex weighted clique algorithm but instead uses a maximum clique algorithm.

3.3 Previous Method

Before presenting our improved clique-based method, we briefly review the previous clique-based method (Fukagawa et al., 2011) (see also Fig. 4), which is referred to as **CliqueEdit** in this paper.

(Figure 4)

CliqueEdit is based on a simple reduction from the tree edit distance problem for unordered trees to the maximum vertex weighted clique problem. Based on Eq. (1), for calculating the tree edit distance, it is enough to find a mapping M maximizing $\sum_{(u,v)\in M} f(u,v)$. In order to find such a mapping, an undirected graph $G = (V, E)$ is constructed from two input trees T_1 and T_2 by

$$\begin{aligned} V &= \{\ (u,v) \mid u \in V(T_1),\ u \neq r(T_1),\ v \in V(T_2),\ v \neq r(T_2)\ \}, \\ E &= \{\ \{(u_1,v_1),(u_2,v_2)\} \mid u_1 \neq u_2,\ v_1 \neq v_2, \\ &\qquad\qquad u_1 \in des(u_2) \text{ iff } v_1 \in des(v_2), \\ &\qquad\qquad u_2 \in des(u_1) \text{ iff } v_2 \in des(v_1)\ \}. \end{aligned}$$

Then, we can see that there is a one-to-one correspondence between the set of cliques and the set of mappings (i.e., (u, v) in a clique corresponds to (u, v) in a mapping M). By assigning a weight $w(x) = f(u, v)$ to each vertex $x = (u, v) \in V$, an optimal mapping M_{OPT} corresponds to a maximum vertex weighted clique. Therefore, the tree edit distance problem can be solved by computing a maximum vertex weighted clique.

3.4 Reduction from the Maximum Vertex Weighted Clique Problem to the Maximum Clique Problem

In order to utilize MCS (Tomita et al., 2010) instead of MWCQ (Nakamura and Tomita, 2005), we develop a simple method that transforms the maximum vertex weighted clique problem into the maximum clique problem.

Let $G = (V, E)$ be a weighted graph such that $V = \{v_1, \ldots, v_n\}$ and each vertex v_i has a weight $w(v_i)$. From $G = (V, E)$, we construct an unweighted graph $\hat{G} = (\hat{V}, \hat{E})$ by (see also Fig. 5)

$$\hat{V} = \{v_i^j \mid v_i \in V,\ j = 1, 2, \ldots, w(v_i)\}, \tag{2}$$

$$\hat{E} = \{\{v_i^j, v_k^\ell\} \mid \{v_i, v_k\} \in E \vee (\ i = k \wedge j \neq \ell\ )\}. \tag{3}$$

(Figure 5)

Proposition 1. *The weight of the maximum vertex weighted clique of $G = (V, E)$ is equal to the size of the maximum clique of $\hat{G} = (\hat{V}, \hat{E})$.*

Proof. Suppose that there exists a clique $G_c = (V_c, E_c)$ with the weight W and the size m in G, where

$$\begin{aligned} V_c &= \{v_{i_1}, \ldots, v_{i_m}\}, \\ E_c &= \binom{V_c}{2}. \end{aligned}$$

In this case, there also exists a clique $\hat{G}_c = (\hat{V}_c, \hat{E}_c)$ in $\hat{G}$, where

$$\begin{aligned} \hat{V}_c &= \{v_{i_k}^j \mid v_{i_k} \in V_c,\ j = 1, 2, \ldots, w(v_{i_k})\}, \\ \hat{E}_c &= \{\{v_{i_h}^j, v_{i_k}^\ell\} \mid \{v_{i_h}, v_{i_k}\} \in E_c \vee (i_h = i_k \wedge j \neq \ell)\}. \end{aligned}$$

Since $|\hat{V}_c| = W$ and $\hat{E}_c = \binom{\hat{V}_c}{2}$, $\hat{G}_c$ is a clique with the size W. Hence, if there exists a clique with the weight W in G, there exists a clique with the size W in $\hat{G}$, so that if G has the maximum vertex weighted clique with the weight W, $\hat{G}$ has the maximum clique with the size W.

Conversely, we assume that there exists a clique $\hat{G}_c = (\hat{V}_c, \hat{E}_c)$ with the size W in $\hat{G}$, where

$$\begin{aligned} \hat{V}_c &= \{v_{i_1}^{j_1}, \ldots, v_{i_W}^{j_W}\}, \\ \hat{E}_c &= \binom{\hat{V}_c}{2}. \end{aligned}$$

Here, $v_{i_k}^{j_k}$ is a copy of v_{i_k} and $i_h = i_k$ can hold. From the way to construct $\hat{G}$ (see Eq. (2) and (3)), there also exists at least one clique $\hat{G}'_c = (\hat{V}'_c, \hat{E}'_c)$ with the size $W'(\geq W)$ in $\hat{G}$, where

$$\begin{aligned} \hat{V}'_c &= \{v_{i_k}^j \mid v_{i_k}^{j_k} \in \hat{V}_c,\ j = 1, \ldots, w(v_{i_k})\}, \\ \hat{E}'_c &= \binom{\hat{V}'_c}{2}. \end{aligned}$$

In this case, there also exists a clique $G_c = (V_c, E_c)$ in G, where

$$\begin{aligned} V_c &= \{v_{i_k} \mid v_{i_k}^{j_k} \in \hat{V}_c\}, \\ E_c &= \{\{v_{i_h}, v_{i_k}\} \mid \{v_{i_h}^j, v_{i_k}^\ell\} \in \hat{E}'_c \wedge i_h \neq i_k\}. \end{aligned}$$

Since the weight of G_c is equal to W' and $E_c = \binom{V_c}{2}$, G_c is a clique with the weight W'. Hence, if there exists a clique with the size W in $\hat{G}$, there exists a clique with the size $W'(\geq W)$ in $\hat{G}$, so that there exist a clique with the weight W' in G. Thus, if $\hat{G}$ has the maximum clique with the size W, G has the maximum vertex weighted clique with the weight W. Therefore, the weight of the maximum vertex weighted clique of G is equal to the size of the maximum clique of $\hat{G}$. □

The method of combining this transformation with CliqueEdit is called **UwCliqueEdit**. Since a vertex with weight w is transformed into w vertices, this method can only be applied to graphs with small integer vertex weights. However, if we consider the unit cost edit distance, each vertex in $G = (V, E)$ has weights 1 or 2. Therefore, this method can be applied to computation of the unit cost tree edit distance.

3.5 Improved Method

In order to improve CliqueEdit, we combine a *dynamic programming* (DP) approach employed in (Akutsu et al., 2011a) with the clique-based approach. We call the resulting method **DpCliqueEdit**.

Let $(u, v) \in V(T_1) \times V(T_2)$. We define $W[u, v]$ be the score of an optimal mapping between $T_1(u)$ and $T_2(v)$ where the root of $T_1(u)$ need not correspond to the root of $T_2(v)$. We compute $W[u, v]$ in a bottom up way (i.e., from leaves to roots) using DP. Suppose that $W[u', v']$ are already computed for all $(u', v') \in des(u) \times des(v)$. Then, we construct an undirected vertex weighted graph $G_{(u,v)} = (V_{(u,v)}, E_{(u,v)})$ by

$$\begin{aligned} V_{(u,v)} &= \{\ (u_1, v_1) \mid u_1 \in des(u),\ v_1 \in des(v)\ \}, \\ E_{(u,v)} &= \{\ \{(u_1, v_1), (u_2, v_2)\} \mid u_1 \neq u_2,\ v_1 \neq v_2, \\ &\qquad u_1 \notin des(u_2),\ u_2 \notin des(u_1), \\ &\qquad v_1 \notin des(v_2),\ v_2 \notin des(v_1)\ \}, \\ w((u_1, v_1)) &= W[u_1, v_1]. \end{aligned}$$

Let W_{max} be the weight of the maximum vertex weight clique for $G_{(u,v)}$. Then, we calculate $W[u, v]$ by[2]

$$W[u, v] = \max \begin{cases} \max_{v' \in des(v)} W[u, v'], \\ \max_{u' \in des(u)} W[u', v], \\ W_{max} + f(u, v), \end{cases}$$

where $W[u, v]$ is initialized by

$$W[u, v] = \begin{cases} \max_{v' \in \{v\} \cup des(v)} f(u, v') \text{ if } u \text{ is a leaf}, \\ \max_{u' \in \{u\} \cup des(u)} f(u', v) \text{ if } v \text{ is a leaf}. \end{cases}$$

Different from the reduction in CliqueEdit, edges are not created in DpCliqueEdit if there is a descendant-ancestor relation between u_1 and u_2 (or between v_1 and v_2, see also Fig. 6). Therefore, it is expected that graphs constructed in DpCliqueEdit are much sparser than those in CliqueEdit though DpCliqueEdit must solve many clique instances. Since sparseness of the graph greatly affects the efficiency of clique finding, it is also expected that DpCliqueEdit is faster than CliqueEdit if non-small trees are given. It is to be noted that transformation to maximum clique cannot be applied to this case because $W[u, v]$ might take a large value even for the unit cost case.

[2] A slight modification is required if u or v is a root because roots cannot be deleted or inserted.

(Figure 6)

3.6 Heuristics

In addition to the use of dynamic programming, we introduce some heuristic techniques to reduce the computation time without violating the optimality of the solution.

An important observation is that

$$W[u_1, v_1] \geq W[u_2, v_1] \tag{4}$$

always holds if u_2 is a descendant of u_1. Based on it, we introduce the following two heuristic techniques.

(1) Each of u and v has only one child.

In this case, we need not construct $G_{(u,v)}$. Instead, we can compute $W[u, v]$ simply by taking the maximum of

$$W[u, v_1],\ W[u_1, v],\ W[u_1, v_1] + f(u, v),$$

where u_1 and v_1 are the children of u and v, respectively (see also Fig. 7).

(Figure 7)

Proposition 2. *If each of $u \in T_1$ and $v \in T_2$ has only one child, $W[u, v]$ can be computed by $W[u, v] = \max\{W[u, v_1], W[u_1, v], W[u_1, v_1] + f(u, v)\}$, where u_1 and v_1 are the children of u and v, respectively.*

Proof. We consider the following three cases.

(i) In the case that u (resp. v) is deleted and v (resp. u) is not deleted, from the assumption and Eq. (4), since $W[u_1, v] \geq W[u'_1, v]$ for all $u'_1 \in des(u_1)$ (resp. $W[u, v_1] \geq W[u, v'_1]$ for all $v'_1 \in des(v_1)$), $W[u, v] = W[u_1, v]$ (resp. $W[u, v] = W[u, v_1]$) holds.

(ii) In the case that u corresponds to v, since each of u and v has only one child, $W_{max} = W[u_1, v_1]$, where W_{max} is the weight of the maximum vertex weight clique for $G_{(u,v)}$. Hence, $W[u, v] = W_{max} + f(u, v) = W[u_1, v_1] + f(u, v)$.

(iii) In the case that both u and v are deleted, $W[u, v] = W[u_1, v_1]$ apparently. Now, the score of this case is smaller than or equal to that of case (ii), because $f(u, v) \geq 0$.

Therefore, we can calculate $W[u, v]$ by

$$W[u,v] \quad = \quad \max \begin{cases} W[u, v_1], \\ W[u_1, v], \\ W[u_1, v_1] + f(u, v). \end{cases}$$

□

(2) $u_2 \in des(u)$ (resp. $v_2 \in des(v)$) does not have a sibling.

In this case, we need not generate a vertex (u_2, v') for any v' (resp. (u', v_2) for any u') in the construction of $G_{(u,v)}$ because any mapping between $T_1(u_2)$ and $T_2(v)$ can be included in some mapping between $T_1(u_1)$ and $T_2(v)$ where u_1 is the parent of u_2 (see also Fig. 8).

(Figure 8)

Proposition 3. *Suppose that $u \in T_1$ and $v \in T_2$. If $u_2 \in des(u)$ (resp. $v_2 \in des(v)$) does not have a sibling, $W[u, v]$ can be computed without generating a vertex (u_2, v') for any v' (resp. (u', v_2) for any u') in the construction of $G_{(u,v)}$, where $v' \in des(v)$ (resp. $u' \in des(u)$).*

Proof. Consider a mapping between $T_1(u)$ and $T_2(v)$. A weighted graph $G_{(u,v)}$ is constructed from $T_1(u)$ and $T_2(v)$. Besides, let $G_{c_2} = (V_{c_2}, E_{c_2})$ be a clique including $(u_2, v') \in V_{(u,v)}(G_{(u,v)})$. Here, because any mapping between $T_1(u_2)$ and $T_2(v)$ can be included in some mapping between $T_1(u_1)$ and $T_2(v)$, there also exists a clique $G_{c_1} = (V_{c_1}, E_{c_1})$ including $(u_1, v') \in V_{(u,v)}(G_{(u,v)})$, where u_1 is the parent of u_2, and $V_{c_2}(G_{c_2}) \setminus \{(u_2, v')\} = V_{c_1}(G_{c_1}) \setminus \{(u_1, v')\}$. Furthermore, $W[u_1, v'] \geq W[u_2, v']$. Let $w(G_{c_1})$ and $w(G_{c_2})$ be the weights of the G_{c_1} and G_{c_2}, respectively. Then, the following inequality holds:

$$\begin{aligned} w(G_{c_1}) &= \sum_{(x,y) \in V_{c_1}} W[x, y] \\ &= \sum_{(x,y) \in V_{c_1} \setminus \{(u_1, v')\}} W[x, y] + W[u_1, v'] \\ &\geq \sum_{(x,y) \in V_{c_2} \setminus \{(u_2, v')\}} W[x, y] + W[u_2, v'] \\ &= \sum_{(x,y) \in V_{c_2}} W[x, y] \\ &= w(G_{c_2}). \end{aligned}$$

Hence, the score of G_{c_2} is smaller than or equal to that of G_{c_1}. Therefore, we need not generate a vertex (u_2, v') for any v' in the construction of $G_{(u,v)}$. □

Although DpCliqueEdit with heuristic techniques (1) and (2) is much faster than CliqueEdit (Akutsu et al., 2011b) in most cases, it takes very long CPU time in some cases, especially if there exist many leaves. In such a case, constructed graphs would contain many vertices and edges and thus a clique algorithm does not work efficiently. In order to cope with such difficult cases, we introduce other heuristic techniques as follows.

The efficiency of MWCQ is much affected by the number of edges in $G_{(u,v)}$. Due to the definition of $E_{(u,v)}$ described in Section 3.5, if u_1, v_1, u_2, v_2 are leaves, there is always an edge between (u_1, v_1) and (u_2, v_2). Therefore, if there are many leaves in T_1 and T_2, $G_{(u,v)}$ has many edges and then MWCQ takes much longer computation time. Since more than a half of nodes are leaves even in binary trees, some heuristic techniques handling leaves are necessary for further speed up.

(3) u_1, u_2, v_1, and v_2 ($u_1 \neq u_2,\ v_1 \neq v_2$) are leaves, and $\ell(u_1) = \ell(u_2)$ or $\ell(v_1) = \ell(v_2)$.

In this case, we need not create an edge $\{(u_1, v_2), (u_2, v_1)\}$ for any u_1, u_2, v_1, and v_2 in the construction of $G_{(u,v)}$ because the score of a mapping including two pairs (u_1, v_2), (u_2, v_1) is equal to that of a mapping including two pairs (u_1, v_1), (u_2, v_2). Therefore, we have only to create an edge $\{(u_1, v_1), (u_2, v_2)\}$ without creating an edge $\{(u_1, v_2), (u_2, v_1)\}$ (see also Fig. 9).

(Figure 9)

Proposition 4. *Suppose that $u_1, u_2 \in T_1$ and $v_1, v_2 \in T_2$. If u_1, u_2, v_1, and v_2 ($u_1 \neq u_2,\ v_1 \neq v_2$) are leaves, and $\ell(u_1) = \ell(u_2)$ or $\ell(v_1) = \ell(v_2)$, $W[u, v]$ can be computed without creating an edge $\{(u_1, v_2), (u_2, v_1)\}$ for any u_1, u_2, v_1, and v_2 in the construction of $G_{(u,v)}$.*

Proof. If there exists a mapping M which includes two pairs (u_1, v_1), (u_2, v_2) between T_1 and T_2, it is implied that there also exists a mapping M' which includes two pairs (u_1, v_2), (u_2, v_1) instead of (u_1, v_1), (u_2, v_2) between T_1 and T_2. Now, the following equality holds:

$$\begin{aligned}
score(M) &= \sum_{(x,y)\in M} W[x,y] \\
&= \sum_{(x,y)\in M\setminus\{(u_1,v_1),(u_2,v_2)\}} W[x,y] + f(u_1,v_1) + f(u_2,v_2) \\
&= \sum_{(x,y)\in M\setminus\{(u_2,v_1),(u_1,v_2)\}} W[x,y] + f(u_2,v_1) + f(u_1,v_2) \\
&= \sum_{(x,y)\in M'} W[x,y] \\
&= score(M')
\end{aligned}$$

Hence, the score of a mapping including two pairs (u_1, v_2), (u_2, v_1) is equal to that of a mapping including two pairs (u_1, v_1), (u_2, v_2). Therefore, we have only to create an edge $\{(u_1, v_1), (u_2, v_2)\}$ without creating an edge $\{(u_1, v_2), (u_2, v_1)\}$. □

(4) u_1, u_2, v_1, and v_2 ($u_1 \neq u_2,\ v_1 \neq v_2$) are leaves, and all labels of them are different.

In this case, we need not create an edge $\{(u_1, v_2), (u_2, v_1)\}$ for any u_1, u_2, v_1, and v_2 in the construction of $G_{(u,v)}$ for the same reason as in the case (3).

Proposition 5. *Suppose that $u_1, u_2 \in T_1$ and $v_1, v_2 \in T_2$. If u_1, u_2, v_1, and v_2 ($u_1 \neq u_2,\ v_1 \neq v_2$) are leaves, and $\ell(u_1), \ell(u_2), \ell(v_1)$, and $\ell(v_2)$ are different, $W[u, v]$ can be computed without creating an edge $\{(u_1, v_2), (u_2, v_1)\}$ for any u_1, u_2, v_1, and v_2 in the construction of $G_{(u,v)}$.*

Proposition 5 can be proved in the same way as the proof of the Proposition 4, so we omit the proof of Proposition 5.

The idea of (3) focusing on the same labeled leaves is extended to the isomorphic subtrees. If T_1 and T_2 are isomorphic including label information, we write $T_1 \approx T_2$.

(5) $T_1(u_1) \approx T_1(u_2)$ ($u_1 \neq u_2$) or $T_2(v_1) \approx T_2(v_2)$ ($v_1 \neq v_2$).

In this case, we need not create an edge $\{(u_1, v_2), (u_2, v_1)\}$ for any u_1, u_2, v_1, and v_2 in the construction of $G_{(u,v)}$ because of the following reason. When $T_1(u_1) \approx T_1(u_2)$ or $T_2(v_1) \approx T_2(v_2)$, the score of mapping $\{(T_1(u_1), T_2(v_2)), (T_1(u_2), T_2(v_1))\}$ is equal to that of mapping $\{(T_1(u_1), T_2(v_1)),(T_1(u_2), T_2(v_2))\}$. Therefore, we have only to create an edge $\{(u_1, v_1), (u_2, v_2)\}$ without creating an edge $\{(u_1, v_2), (u_2, v_1)\}$ (see also Fig. 10).

(Figure 10)

Proposition 6. *Suppose that $u_1, u_2 \in T_1$ and $v_1, v_2 \in T_2$. If $T_1(u_1) \approx T_1(u_2)$ ($u_1 \neq u_2$) or $T_2(v_1) \approx T_2(v_2)$ ($v_1 \neq v_2$), $W[u, v]$ can be computed without creating an edge $\{(u_1, v_2), (u_2, v_1)\}$ for any u_1, u_2, v_1, and v_2 in the construction of $G_{(u,v)}$.*

Proof. If there exists a mapping M which includes two pairs (u_1, v_1), (u_2, v_2) between T_1 and T_2, it is implied that there also exists a mapping M' which includes two pairs (u_1, v_2), (u_2, v_1) instead of (u_1, v_1), (u_2, v_2) between T_1 and T_2. Moreover, from the assumption, the score of mapping $\{(T_1(u_1), T_2(v_2)), (T_1(u_2), T_2(v_1))\}$ is equal to that of mapping $\{(T_1(u_1), T_2(v_1)),\ (T_1(u_2), T_2(v_2))\}$, that is, $W[u_1, v_1] = W[u_2, v_1]$ and $W[u_2, v_2] = W[u_1, v_2]$. Now, the following equality holds,

$$
\begin{aligned}
score(M) &= \sum_{(x,y)\in M} W[x, y] \\
&= \sum_{(x,y)\in M\setminus\{(u_1,v_1),(u_2,v_2)\}} W[x, y] + W[u_1, v_1] + W[u_2, v_2] \\
&= \sum_{(x,y)\in M\setminus\{(u_2,v_1),(u_1,v_2)\}} W[x, y] + W[u_2, v_1] + W[u_1, v_2] \\
&= \sum_{(x,y)\in M'} W[x, y] \\
&= score(M')
\end{aligned}
$$

Thus, the score of a mapping including two pairs (u_1, v_2), (u_2, v_1) is equal to that of a mapping including two pairs (u_1, v_1), (u_2, v_2). Therefore, we have only to create an edge $\{(u_1, v_1), (u_2, v_2)\}$ without creating an edge $\{(u_1, v_2), (u_2, v_1)\}$. □

It is expensive to determine whether two graphs are isomorphic or not but we can solve the problem easier when the two graphs are trees. Though various algorithms are invented for the problem, we employ an algorithm introduced in (Matoušek and Nešetřil, 1998). The algorithm transforms the tree isomorphism problem into the comparison of simple numerical sequences.

From the propositions $2 \sim 6$, we have the following theorem.

Theorem 1. *DpCliqueEdit with the heuristic* $(1) \sim (5)$ *computes* $dist(T_1, T_2)$ *without violating the optimality of the solution.*

It should be noted that we can use the heuristic techniques only if DP is introduced to the clique-based approach.

4 EXPERIMENTAL RESULTS

In order to evaluate the efficiency of the improved method and heuristic techniques, we applied CliqueEdit, UwCliqueEdit, and DpCliqueEdit to comparison of real tree structured data. As the tree structured data, we employed glycan structures that were obtained from KEGG/Glycan database (Kanehisa et al., 2010) and CSLOGS dataset which consists of Web logs files (Zaki et al., 2005).

It is to be noted that, as far as we know, there exists no other publicly available program for exactly computing the unordered tree edit distance and thus we only compared these methods. From the result given in (Horesh et al., 2006), it is considered that CliqueEdit has similar efficiency (Fukagawa et al., 2011) to the A^*-algorithm for unordered tree edit distance (Horesh et al., 2006).

We implemented CliqueEdit, UwCliqueEdit, and DpCliqueEdit using C++ language and compared UwCliqueEdit and DpCliqueEdit with the previous method CliqueEdit. In the implementations of CliqueEdit and DpCliqueEdit, MWCQ (Nakamura and Tomita, 2005) was used as a maximum vertex weighted clique algorithm, while in that of UwCliqueEdit, MCS (Tomita et al., 2010) was used as a maximum clique algorithm. **DpCliqueEdit-A**, **DpCliqueEdit-B**, **DpCliqueEdit-C**, **DpCliqueEdit-D**, and **DpCliqueEdit-E** represent DpCliqueEdit without heuristics, with heuristics (1)(2), with heuristics (1)(2)(3), with heuristics (1)(2)(3)(5), and with all heuristics, respectively. The preliminary version of DpCliqueEdit in (Akutsu et al., 2011b) is equivalent to DpCliqueEdit-B. We performed computational experiments using a PC with 2.66 GHz Intel Core i7 CPU and 3.88 GB RAM running under the Mac OS X operating system. In this paper, we focus only on the computational efficiency and do not conduct computational experiments for evaluating the performance (i.e., accuracy of comparison) of CliqueEdit, UwCliqueEdit, and DpCliqueEdit because these methods compute the same distances, the performance of CliqueEdit was already evaluated in the previous work (Fukagawa et al., 2011), and the tree edit distance is the most established distance measure for trees (Bille, 2005).

For evaluation of the methods, we used the standard weighting scheme (i.e., $f(u, v) = 2$ for $\ell(u) = \ell(v)$, $f(u, v) = 1$ for $\ell(u) \neq \ell(v)$) corresponding to the unit cost edit distance.

4.1 Glycan Structures

As in our previous work (Fukagawa et al., 2011), we randomly selected 100 pairs of glycan structures with a specified range of the total number of nodes (i.e., the sum of the numbers of nodes in T_1 and T_2) and measured the average CPU time (user time) per pair. Unbalanced cases in which the size of one structure was smaller than 1/3 of the other structure were excluded. For each of the ranges in $60 \sim 79$, we took the average over 20 pairs because there did not exist an enough number of pairs, where we could use 19 pairs among 20 pairs for the range of $70 \sim 74$ because there was a hard case for which DpCliqueEdit-A could not output a solution within 60 minutes. For the ranges of $80 \sim 84$, $85 \sim 89$, and $90 \sim 94$, only 9, 5, and 4 pairs were available, respectively. We could use only 4 pairs among 5 pairs for the range of $85 \sim 89$ and 2 pairs among 4 pairs

for the range of $90 \sim 94$ because there were hard cases for which DpCliqueEdit-A could not output a solution within 60 minutes.

The result of the computational experiment is shown in Table 1. From this table, it is seen that DpCliqueEdit-B, DpCliqueEdit-C, DpCliqueEdit-D, and DpCliqueEdit-E are much faster than CliqueEdit, UwCliqueEdit, and DpCliqueEdit-A for non-small glycan structures. In particular, DpCliqueEdit-D is the fastest for comparison of large glycan structures. Although UwCliqueEdit is faster than CliqueEdit in most cases, it is not fast for comparison of large glycan structures because it constructs larger and denser graph $\hat{G}$ as G becomes larger and thus MCS does not work efficiently. Besides, DpCliqueEdit-A is not fast despite the fact that DpCliqueEdit-B $\sim$ DpCliqueEdit-E are faster than the other methods. This is because DpCliqueEdit repeatedly solves instances of the maximum vertex weighted clique problem as sub-problems, so that it takes long CPU time if the heuristic techniques are not introduced. Since the heuristic techniques proposed in this paper cannot be used without using DP, DP needs to be introduced in order to reduce the computation time. Although CliqueEdit and UwCliqueEdit are faster than DpCliqueEdit-B $\sim$ DpCliqueEdit-E for small glycan structures, comparison of large glycan structures is more crucial because it takes a large amount of time.

(Table 1)

Table 2 shows the results on pairs of trees (i.e., hard cases) for which some of the examined methods could not compute the distance within 60 minutes. From this table, though there is no great difference between DpCliqueEdit-B $\sim$ DpCliqueEdit-E except for the range of $90 \sim 94$ in Table 1, we find that DpCliqueEdit-D and DpCliqueEdit-E are much faster than the other methods in hard cases. This implies that DpCliqueEdit-D and DpCliqueEdit-E utilize the existence of the same labeled leaves, different labeled leaves, and isomorphic subtrees, and thus need much shorter time for MWCQ. It takes long CPU time for DpCliqueEdit-A to output a solution in most cases. This is because it costs much CPU time to construct $G_{(u,v)}$ and solve the maximum vertex weighted clique problem repeatedly. It is also seen that there exist some instances which UwCliqueEdit can solve within 60 minutes whereas CliqueEdit or DpCliqueEdit-B cannot solve within 60 minutes. However, UwCliqueEdit is not faster than CliqueEdit and DpCliqueEdit-B for comparison of large glycan structures.

From the results of these computational experiments, we can conclude that DpCliqueEdit-D and DpCliqueEdit-E are more useful than the other proposed clique-based methods. Using heuristics (3) and (5), DpCliqueEdit-D and DpCliqueEdit-E are much faster than DpCliqueEdit-B (i.e., the preliminary version of DpCliqueEdit in (Akutsu et al., 2011b)).

(Table 2)

4.2 CSLOGS Dataset

As in the comparison of glycan structures, we randomly selected 100 pairs of Web logs data with a specified range of the total number of nodes (i.e., the sum of the numbers of

nodes in T_1 and T_2) and measured the average CPU time (user time) per pair. Different from comparison of the glycan structures, we randomly selected trees from CSLOGS and created two sub-datasets called SUBLOGS3 and SUBLOGS5 in this paper. Each sub-dataset has 15000 trees whose sizes are restricted to smaller than or equal to 80, where the maximum number of children of each node is limited to smaller than or equal to 3 and 5 in SUBLOGS3 and SUBLOGS5, respectively. The percentage of the number of trees in which the maximum number of children of each node are restricted to 3 and 5 is about 65% and 81% of the total number of them in CSLOGS. Unbalanced cases mentioned in Section 4.1 were excluded.

The results of the computational experiments we performed with SUBLOGS3 and SUBLOGS5 are shown in Table 3 and Table 4, respectively. From Table 3, it is seen that UwCliqueEdit is the fastest for small trees. However, as the total number of nodes of input trees becomes larger, it takes longer CPU time for UwCliqueEdit to solve the problem, and there exist hard cases for which UwCliqueEdit could not output a solution within 60 minutes. For non-small trees, although most methods could not solve the problem in 60 minutes in some cases, DpCliqueEdit-E could in 60 minutes for all cases we selected in this experiment. Similarly, from Table 4, we find that UwCliqueEdit is faster than any other method for small trees and DpCliqueEdit-E is the fastest for non-small trees.

Although there is no great difference between DpCliqueEdit-D and DpCliqueEdit-E for comparison of glycan structures, DpCliqueEdit-E is the most useful for comparison of trees. In CSLOGS, there are 13,361 unique Web page (Zaki et al., 2005) and each Web page is assigned to each node as a label, so that there exist many leaves with different labels. Therefore, heuristic (4) works efficiently.

(Table 3)

(Table 4)

5 CONCLUSION

In this paper, we proposed an improved clique-based method by introducing DP and several heuristic techniques for computing the tree edit distance between rooted unordered trees. DP and the heuristic techniques are very useful and then the improved method is much faster than the previous method in most cases of comparison of real tree structured data. In particular, for hard instances of comparison of glycan structures, the improved method achieved more than 100 times speed-up. Although the improved method is not faster for comparison of small glycans, it is not crucial because comparison of large glycan structures takes much longer CPU time than that of small glycans. In comparison of Web logs data, it takes long CPU time to compute the edit distance between trees when there exist some nodes with many children. However, most biological data such as glycans, RNA secondary structures, and vascular trees might have few internal nodes with many children.

Although the improved method is much faster than the previous method, there still exist cases for which it takes long CPU time. In particular, it takes long CPU time if there exist long subtrees (i.e., there exist many nodes but few leaves) because the heuristics (1)(2) proposed in this paper can reduce the computation time only if there exist nodes with one child in the long subtrees, and the heuristics (3)(4)(5) cannot well contribute to reduction of the number of edges in such cases and thus the maximum vertex weighted clique algorithm does not work efficiently. How to cope with such cases is left for future work.

Moreover, in order to achieve further speed-up, we should develop an improved algorithm for the maximum vertex weighted clique problem because an improvement of the efficiency of clique finding directly leads to an improvement of the efficiency of our proposed algorithm. In particular, a maximum vertex weighted clique solver specialized for properties of weighted graphs generated by the clique-based algorithm might be useful for the tree edit distance problem. How to develop such an algorithm is also left as future work.

In addition to the future work mentioned above, some modifications are needed for application to analysis of tree structured data used in computational biology. Although we have used the unit cost edit distance in computational experiments, more suitable cost functions should be used for analysis of biological and other objects. Development of cost functions suitable to individual applications is also left for future work.

ACKNOWLEDGMENTS

This work was partially supported by research collaboration projects by National Institute for Informatics, and Institute for Chemical Research, Kyoto University. The work of TA was partially supported by MEXT Grant-in-Aid No.22240009. The work of DF and AT was partly supported by MEXT Grant-in-Aid No.18049069. The work of ET was partially supported by MEXT Grant-in-Aid No.22500009.

References

Aoki, K. F., Yamaguchi, A., Ueda, N., Akutsu, T., Mamitsuka, H., Goto, S., and Kanehisa, M. 2004. KCaM (KEGG Carbohydrate Matcher): a software tool for analyzing the structures of carbohydrate sugar chains, *Nucleic Asids Research* 32, 267-272.

Akutsu, T., Fukagawa, D., Takasu, A., and Tamura, T. 2011a. Exact algorithms for computing tree edit distance between unordered trees, *Theoretical Computer Science* 421, 352-364.

Akutsu, T., Mori, T., Tamura, T., Fukagawa, D., Talasu, A., and Tomita, E. 2011b An improved clique-based method for computing edit distance between unordered trees and its application to comparison of glycan structures, *The 4th International Workshop on Intelligent Informatics in Biology and Medicine, A Part of Proc. 5th International Conference on Complex, Intelligent and Software Intensive Systems*, 536-540.

Bille, P. 2005. A survey on tree edit distance and related problems, *Theoretical Computer Science* 337, 217-239.

Fukagawa, D., Tamura, T., Takasu, A., Tomita, E., and Akutsu, T. 2011. A clique-based method for the edit distance between unordered trees and its application to analysis of glycan structures, *BMC Bioinformatics* (Suppl. for APBC 2011), 12, Suppl 1, S14 (9 pages).

Demaine, E. D., Mozes, S., Rossman, and B., Weimann, O. 2009. An optimal decomposition algorithm for tree edit distance, *ACM Transactions on Algorithms* 6.

Horesh, Y., Mehr, R., and Unger, R. 2006. Designing an A* algorithm for calculating edit distance between rooted-unordered trees, *Journal of Computational Biology* 13, 1165-1176.

Jiang, T., Lin, G., Ma, B., and Zhang, K. 2002. A general edit distance between RNA structures, 2002. *Journal of Computational Biology* 9, 371-388.

Kanehisa, M., Goto, S., Furumichi, F., Tanabe, M., and Hirakawa, M. 2010. KEGG for representation and analysis of molecular networks, *Nucleic Acids Research* 38, D355-D360.

Matoušek, J., and Nešetřil, J. 1998. Invitation to Discrete Mathematics, Oxford University Press, New York.

Nakamura, T., and Tomita, E. 2005. Efficient algorithms for finding a maximum clique with maximum vertex weight. Technical Report UEC-TR-CAS3-2005 (in Japanese), the University of Electro-Communications.

Ogawa, H. 1986. Labeled point pattern matching by Delaunay triangulation and maximal cliques, *Pattern Recognition*, 35-40.

Pelillo, M., Siddiqi, K., and Zucker, S. W. 1999. Matching hierarchical structures using association graphs, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21, 1105-1119.

Tai, K.-C. 1979. The tree-to-tree correction problem, *Journal of ACM* 26, 422-433.

Tomita, E., and Seki, T. 2003. An efficient branch-and-bound algorithm for finding a maximum clique, *in Proc. 4th International Conference on Discrete Mathematics and Theoretical Computer Science* (Lecture Notes in Computer Science Vol.2731), 278-289.

Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., and Wakatsuki, M. 2010. A simple and faster branch-and-bound algorithm for finding a maximum clique, *In Proc. 4th International Workshop on Algorithms and Computation* (Lecture Notes in Computer Science Vol. 5942), 191-203.

Tomita, E., Akutsu. T., and Matsunaga, T. 2011. Efficient algorithms for finding maximum and maximal cliques: Effective tools for bioinformatics, 625-640. *in* Biomedical Engineering, Trends in Electronics, Communications and Software, Laskovski, A. B. eds., ISBN: 978-953-307-475-7, InTech. Available from: http://www.intechopen.com/articles/show/title/efficient-algorithms-for-finding-maximum-and-maximal-cliques-effective-tools-for-bioinformatics

Torsello, A., and Hancock, E. R. 2003. Computing approximate tree edit distance using relaxation labeling, *Pattern Recognition Letters* 24, 1089-1097.

Yu, K.-C., Ritman, and E. L., Higgns, E. 2007. System for the analysis and visualization of large 3D anatomical trees, *Computers in Biology and Medicine* 37, 1802-1830.

Zaki, M.J. 2005. Efficiently Mining Frequent Trees in a Forest: Algorithms and Applications, *IEEE Transactions on Knowledge and Data Engineering*, special issue on Mining Biological Data, 17, 8, 1021-1035.

Zhang, K., Statman, R., and Shasha, D. 1992. On the editing distance between unordered labeled trees, *Information Processing Letters* 42, 133-139.

Table 1: CPU time for comparing glycans.

total # nodes	CliqueEdit	UwCliqueEdit	DpCliqueEdit-A	DpCliqueEdit-B	DpCliqueEdit-C	DpCliqueEdit-D	DpCliqueEdit-E
$30 \sim 34$	**0.002**	0.003	0.013	0.006	0.006	0.006	0.009
$35 \sim 39$	**0.004**	0.007	0.027	0.011	0.011	0.012	0.017
$40 \sim 44$	0.056	0.035	0.107	0.026	**0.019**	0.021	0.029
$45 \sim 49$	0.064	0.036	0.126	0.031	**0.030**	0.031	0.040
$50 \sim 54$	0.078	0.049	0.228	0.039	**0.037**	0.039	0.051
$55 \sim 59$	1.987	0.433	8.968	0.108	0.088	**0.086**	0.096
$60 \sim 64$	2.746	4.949	1.780	0.167	0.163	**0.149**	0.177
$65 \sim 69$	64.290	9.303	39.460	0.381	0.364	**0.328**	0.357
$70 \sim 74$	58.690	0.099	1.337	0.545	**0.436**	0.463	0.501
$75 \sim 79$	2.441	0.918	4.051	0.953	**0.752**	0.754	0.781
$80 \sim 84$	7.150	6.570	44.630	2.516	2.268	**1.620**	1.653
$85 \sim 89$	237.700	28.030	21.110	3.205	3.205	**2.413**	2.490
$90 \sim 94$	303.200	1211.000	1710.000	38.810	26.300	**8.165**	9.475

Average CPU time (sec.) per glycan pair is shown for each case. Bold face indicates the best results for each case.

Table 2: CPU time for comparing glycans for each hard case.

glycan pair	total # nodes	CliqueEdit	UwCliqueEdit	DpCliqueEdit-A	DpCliqueEdit-B	DpCliqueEdit-C	DpCliqueEdit-D	DpCliqueEdit-E
{G04520, G04682}	35	693.400	-	223.900	225.800	**0.020**	**0.020**	**0.020**
{G04520, G05248}	36	1124.000	-	284.500	285.900	**0.020**	**0.020**	**0.020**
{G03769, G04682}	71	-	491.400	-	-	10.910	**0.490**	0.520
{G03769, G04520}	72	-	59.080	-	-	11.800	**0.420**	0.450
{G03769, G05248}	72	-	17.380	-	-	56.500	**0.600**	0.630
{G03769, G05297}	72	-	17.590	-	-	56.560	**0.600**	0.630
{G03655, G03769}	88	108.600	277.400	-	300.700	31.170	**5.610**	6.430
{G03769, G04206}	91	844.100	1397.000	-	5.870	5.250	5.830	**5.120**
{G03769, G11847}	91	132.100	911.500	-	108.400	82.880	28.120	**22.200**

Average CPU time (sec.) per glycan pair is shown for each case. CPU time (sec.) per glycan pair is shown for each hard case. "-" denotes that the program could not output a solution within 60 minutes (= 3600 seconds). Bold face indicates the best results for each case.

Table 3: CPU time for comparing Web logs data obtained from SUBLOGS3 dataset.

total # nodes	CliqueEdit	UwCliqueEdit	DpCliqueEdit-A	DpCliqueEdit-B	DpCliqueEdit-C	DpCliqueEdit-D	DpCliqueEdit-E
$30 \sim 34$	**0.003**	**0.003**	0.010	0.007	0.007	0.007	0.008
$35 \sim 39$	**0.009**	**0.009**	0.024	0.018	0.017	0.018	0.018
$40 \sim 44$	0.061	**0.023**	0.068	0.038	0.032	0.031	0.033
$45 \sim 49$	0.218	0.100	0.155	0.059	0.053	0.051	**0.050**
$50 \sim 54$	2.928	0.129	0.370	0.222	0.119	0.117	**0.095**
$55 \sim 59$	2.189	0.809	2.965	0.210	0.173	0.182	**0.167**
$60 \sim 64$	-	39.940	-	20.450	0.542	1.904	**0.297**
$65 \sim 69$	-	17.380	-	-	2.230	1.106	**0.662**
$70 \sim 74$	-	-	-	-	3.589	1.423	**1.024**
$75 \sim 79$	-	-	-	-	-	1.895	**1.566**
$80 \sim 84$	-	-	-	-	-	-	**2.625**
$85 \sim 89$	-	-	-	-	-	46.920	**10.550**
$90 \sim 94$	-	-	-	-	-	-	**50.570**
$95 \sim 99$	-	-	-	-	-	-	**64.980**

Average CPU time (sec.) per Web logs pair is shown for each case. The maximum number of children of each node is limited to smaller than or equal to 3. "-" denotes that there exist at least one hard case for which the program could not output a solution within 60 minutes (= 3600 seconds). Bold face indicates the best results for each case.

Table 4: CPU time for comparing Web logs data obtained from SUBLOGS5 dataset.

total # nodes	CliqueEdit	UwCliqueEdit	DpCliqueEdit-A	DpCliqueEdit-B	DpCliqueEdit-C	DpCliqueEdit-D	DpCliqueEdit-E
$30 \sim 34$	0.010	**0.005**	0.029	0.011	0.010	0.011	0.008
$35 \sim 39$	0.244	0.034	0.357	0.114	0.027	0.025	**0.016**
$40 \sim 44$	44.090	4.067	41.000	4.630	2.890	1.913	**0.028**
$45 \sim 49$	35.380	7.310	19.140	3.075	1.079	0.994	**0.101**
$50 \sim 54$	-	-	-	-	-	-	**0.141**
$55 \sim 59$	-	53.750	-	-	12.580	12.260	**0.423**
$60 \sim 64$	-	-	-	-	-	-	**17.240**
$65 \sim 69$	-	-	-	-	-	-	**10.850**

Average CPU time (sec.) per Web logs pair is shown for each case. The maximum number of children of each node is limited to smaller than or equal to 5. "-" denotes that there exist at least one hard case for which the program could not output a solution within 60 minutes (= 3600 seconds). Bold face indicates the best results for each case.

Figures

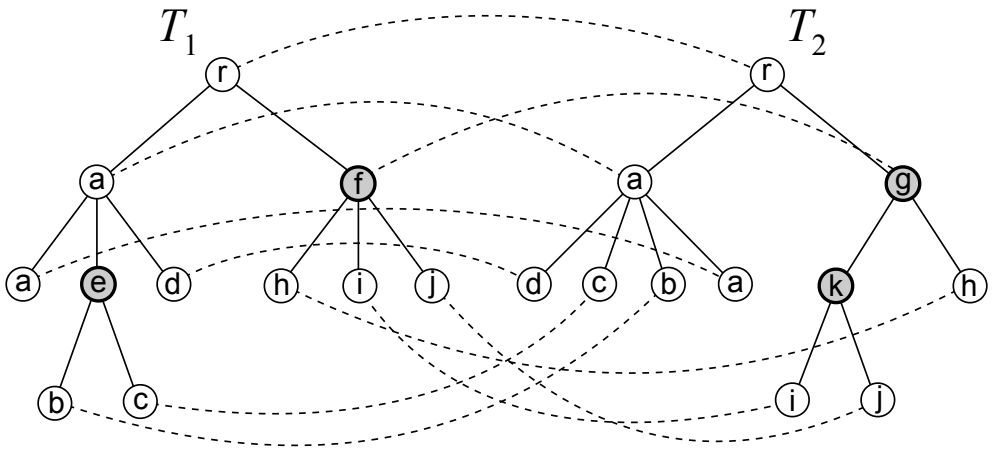
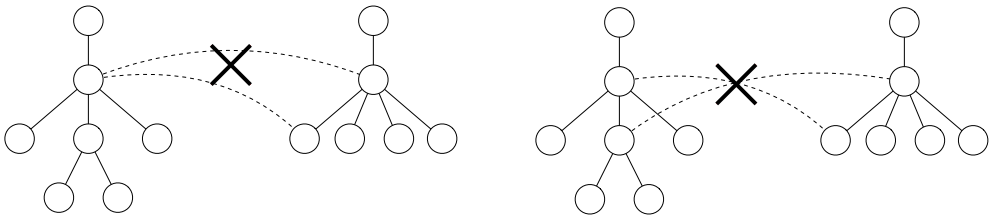


Figure 1

Figure 1: Example of tree edit operations and edit distance mapping for unordered trees. T_2 is obtained from T_1 by deletion of node labeled **e**, insertion of node labeled **k**, and substitution of node labeled **f** with node labeled **g**, where a tree can contain nodes with the same label. The corresponding mapping M is shown by broken curves.



(i) one-to-one condition (ii) descendant condition

Figure 2

Figure 2: Example of the condition of the edit distance mapping. The left and right figures correspond to conditions (i) and (ii) for the edit distance mapping, respectively, where (i) stands for one-to-one condition, and (ii) stands for descendant condition.

(i) maximum clique (ii) maximum vertex weighted clique

Figure 3

Figure 3: Example of the maximum clique and the maximum vertex weighted clique. The size of the maximum clique of the left graph is four, while the weight of the maximum vertex weighted clique of the right graph is 10.

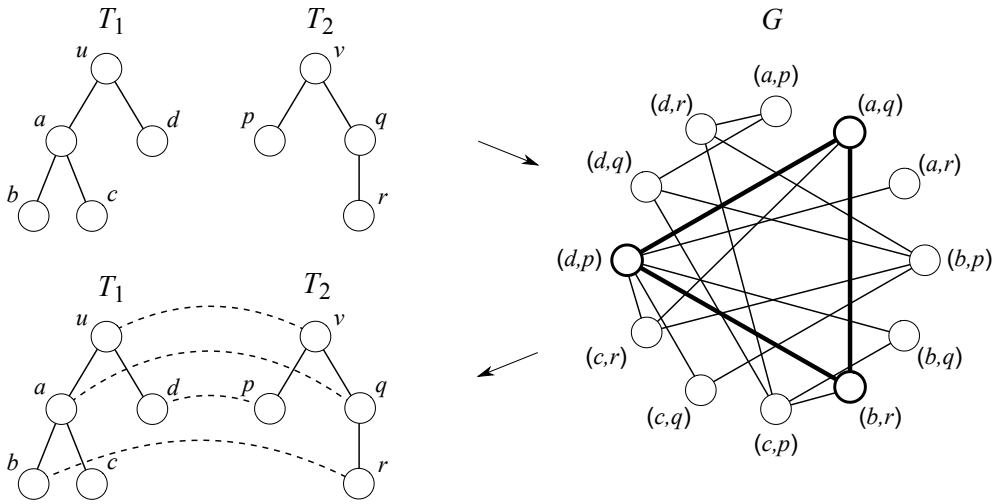


Figure 4

Figure 4: Example of a reduction in the previous method (CliqueEdit). A maximum clique is shown by bold lines in graph G, and the corresponding mapping is shown by broken lines in bottom left side.

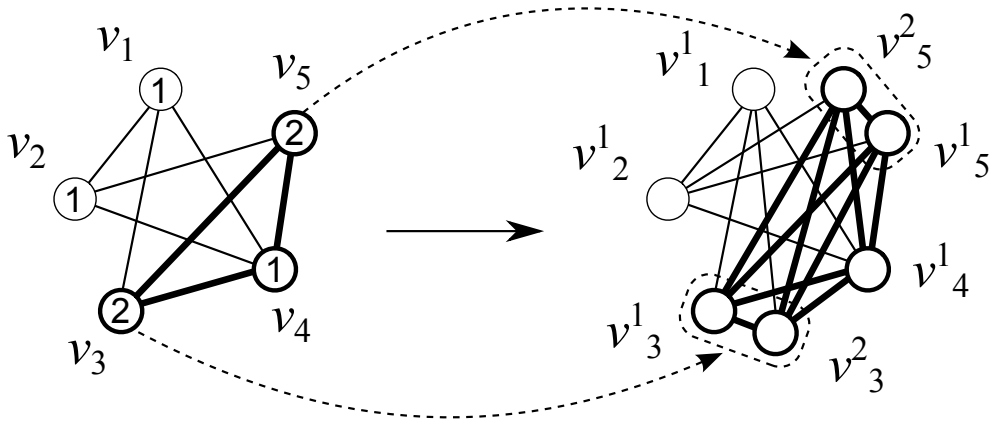


Figure 5

Figure 5: Example of transformation of a vertex weighted graph into an unweighted graph. v_3 and v_5 are duplicated.

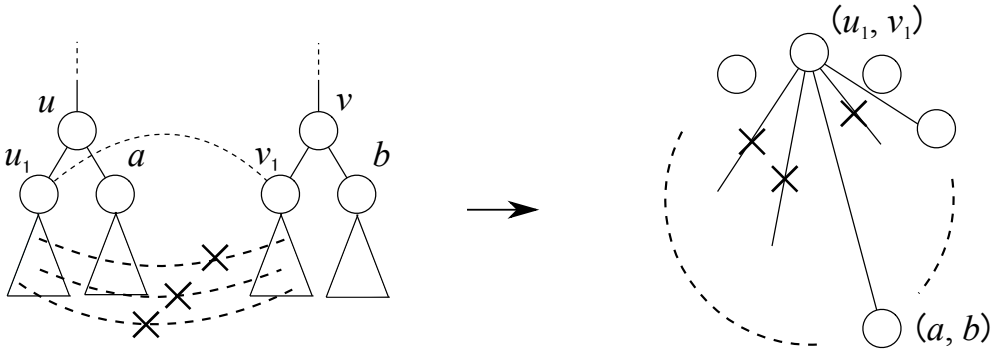


Figure 6

Figure 6: Difference between the reductions in CliqueEdit and DpCliqueEdit. In computation of $W[u, v]$ in DpCliqueEdit, vertex (u_1, v_1) in $G_{(u,v)}$ is not connected to any one of vertices corresponding to pairs of descendants of u_1 and v_1.

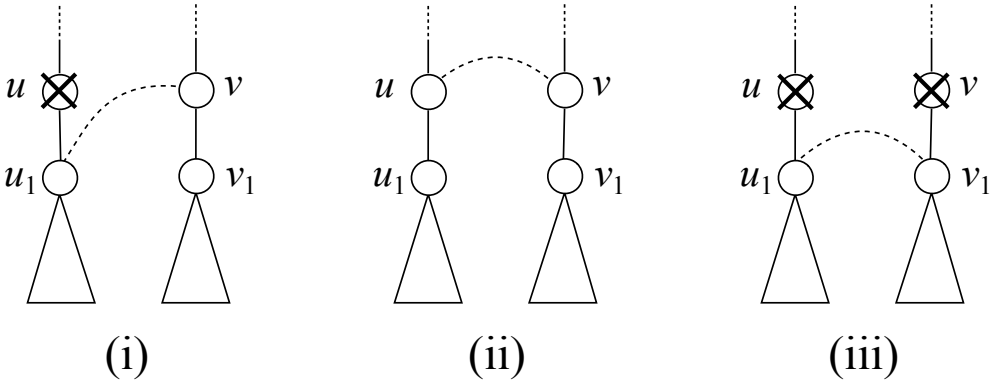


Figure 7

Figure 7: Heuristic techniques (1). (i), (ii), and (iii) denote the cases (i) $\sim$ (iii) in the proof of the Proposition 2.

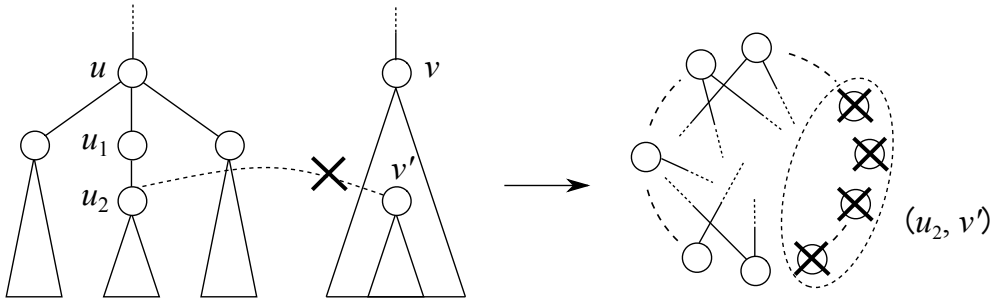


Figure 8

Figure 8: Heuristic techniques (2). A vertex (u_2, v') is not necessary for any v' in the construction of $G_{(u,v)}$.

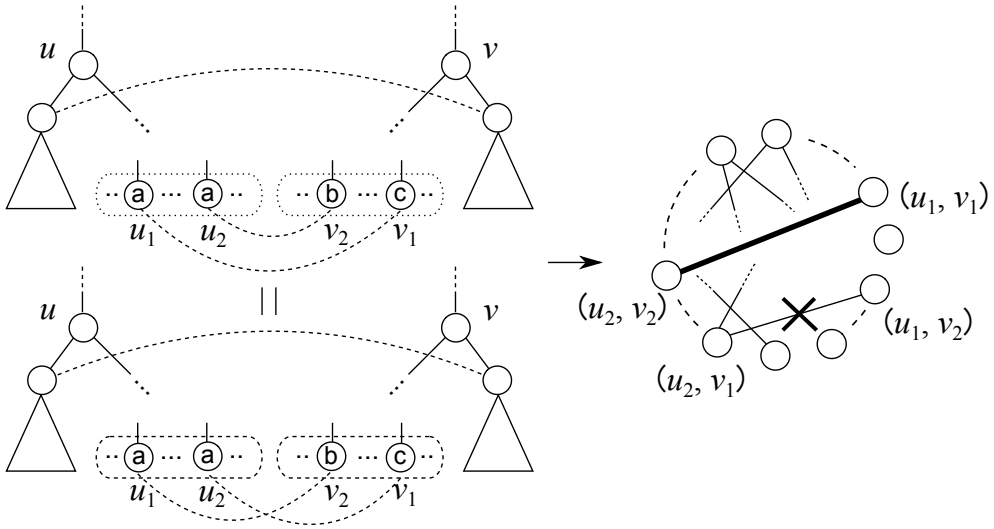


Figure 9

Figure 9: Example of heuristic technique (3). u_1, u_2, v_1, and v_2 are leaves, and $\ell(u_1) = \ell(u_2)$. In this case, we need not create $\{(u_1, v_2), (u_2, v_1)\}$.

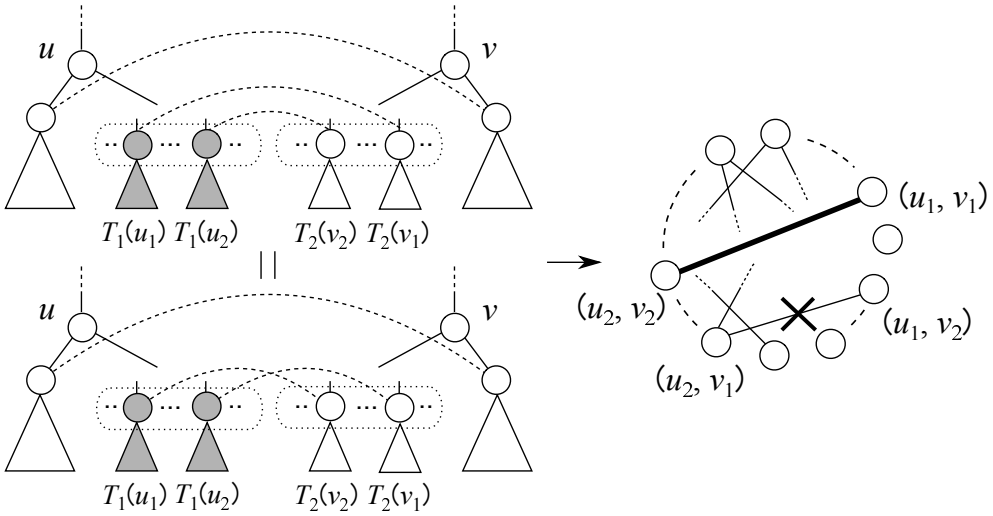


Figure 10

Figure 10: Example of heuristic technique (4). $T_1(u_1)$ and $T_1(u_2)$ are isomorphic including label information. In this case, we need not create $\{(u_1, v_2), (u_2, v_1)\}$.